# Towards a control-theory approach for minimizing unused grid resources

Emmanuel Stahl [2], Agustín Gabriel Yabo [1], Olivier Richard [1], Bruno Bzeznik [1], Bogdan Robu [2], Eric Rutten [1]
[1] Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble France
{firstname.lastname}@inria.fr
[2]Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-lab, F-38000 Grenoble France
bogdan.robu@gipsa-lab.grenoble-inp.fr

***Abstract*—HPC systems are facing more and more variability in their behavior, related to e.g., performance and power consumption, and the fact that they are less predictable requires more runtime management. This can be done in an Autonomic Management feedback loop, in response to monitored information in the systems, by analysis of this data and utilization of the results in order to activate appropriate system-level or application-level feedback mechanisms (e.g., informing schedulers, down-clocking CPUs).**

**One such problem is found in the context of CiGri, a simple, lightweight, scalable and fault tolerant grid system which exploits the unused resources of a set of computing clusters. Computing power left over by the execution of a main HPC application scheduling is used to execute smaller jobs, which are injected as much as the global system allows.**

**This paper presents first results addressing the problem of automated resource management in an HPC infrastructure, using techniques from Control Theory to design a controller that maximizes cluster utilization while avoiding overload. We put in place a mechanism for feedback (Proportional Integral, PI) control system software, through a maximum number of jobs to be sent to the cluster, in response to system information about the current number of jobs processed.**

**Keywords:** High performance computing, resource management, self-adaptive systems, autonomic computing, control theory

## I. Introduction

### A. Need for autonomic administration in HPC

HPC systems are facing more and more variability in their behavior, related to e.g. performance and power consumption, and the fact that they are less predictable requires more runtime management. Some examples in HPC can be:

- uncertainties concerning actual runtime performances (e.g. execution times) compared to evaluations which were used for off-line scheduling.
- variations in data access times due to e.g. more and more elaborate cache mechanisms.
- variations related to the values in the data e.g. number of iterations or depth of computation depending on variables' precision.

Coping with such phenomena must be done at run time, based upon measurements performed during the execution. This can be done in response to monitored information in the systems, by analysis of this data and utilization of the results in order to activate appropriate system-level or application-level feedback mechanisms (e.g. informing schedulers, down-clocking CPUs).

One such problem is found in the context of CiGri, a simple, lightweight, scalable and fault tolerant grid system which exploits the unused resources of a set of computing clusters. Computing power left over by the execution of a main HPC application scheduling is used to execute smaller jobs, which are injected as much as the global system allows.

### B. Control theory for Autonomic HPC

Such feedback mechanisms rely on extensive monitoring and analysis, and involve decisions and their execution. These feedback loops, in the domain of Computer Science, are the object of Autonomic Computing [1], which emerged mainly from distributed and Cloud systems at IBM.

One approach in designing feedback loops is naturally Control Theory, which is extremely widespread in all domains of engineering, but only quite recently and scarcely applied to regulation in computing systems [2], [3]. It can bring to systems designers methodologies to conceive and implement feedback loops with well-mastered and guaranteed behavior in order to obtain automated management with goals of optimization of resources or avoidance of crashes and overloads.

Different control designs have been proposed in the past, aiming to solve different classes of control problems and involving different classes of dynamical models. Classical control addresses quantitative dynamics, using continuous models based on differential equations. Discrete Event Systems concern systems where the dynamics involves a set of discrete, logical states (possibly finite), and events causing transitions: they are commonly modelled using Petri nets or finite state automata. Probabilistic aspects can be addressed with e.g., Markov Decision Processes. The control problems encountered in complex computing systems can be of various kinds, and represent a new application domain for Control Theory, for which research is still identifying and evaluating modelling approaches.

### C. Contributions

This paper presents first results addressing automated resource management using Control Theory. More precisely,

it attacks the regulation of the injection of jobs into a grid system, in such a way that it maximizes its utilization, by leaving only a minimum of resources unsused, while at the same time avoiding to overshoot the bounds leading to a system overload, and hence avoiding the firing of costly safety mechanisms.

Our method proceeds with the following phases:

1) analysis of the system and identification of its dynamics and the overload problems to be tackled.
2) design of a feedback controller, with P (Proportional) and PI (Proportional+Intregral) regulation, through a maximum number of jobs to be sent to the cluster, in response to the current number of jobs processed. The control objective is to maximize cluster utilization while avoiding overload.
3) implementation and experimental results, in an emulated environment, comparing the new control scheme with the previously used, more *ad hoc* solution.

In the remainder of this paper, background is given in Section II on Autonomic Computing and its control, and on the overall Architecture of the system to be controlled, and its current, ad-hoc regulation mechanism. Analysis and modeling of the system, leading to the design of a first proportional controller, and then a PI controller, is described in Section III. Then Section IV shows experimental validation of our approach. Finally, Section V concludes and draws perspectives.

## II. Background

### A. Autonomic Computing and its control

*1) Autonomic administration:* Autonomic Computing (AC) [1] aims at making computing systems able of self-management, w.r.t. self-configuration (deployment), self-optimization (resource management), self-healing (fault tolerance) and self-protection (security). The management of system administration is automated so as to replace, as much as possible, human intervention, for motivations of facing complexity or fast response. AC has been proposed in distributed systems and particularly in the area of Cloud computing. Recently, the perspectives of self-adaptive systems and autonomic management have also been considered in HPC. AC relies on a structure of feedback loop, where a controller, separate from the system to be managed, runs in parallel to it: the well known MAPE-K loop [4] involves components to Monitor (through probes and sensors), Analyze (extracting relevant information from data, and take decisions), Plan (transform decisions into adaptation actions) and Execute (perform adaptation actions). All this is done using Knowledge on the system (reified representation, data-base).

*2) Control-based approaches:* The classical MAPE-K AC loop can be considered in the perspective of Control Theory [4], and its feedback loop represented in Figure 1. In this domain, a deep and long history of research and practice has been developed, applied to most domains of engineering, but only quite recently to management issues in computing systems [2], [3], [5], [6].

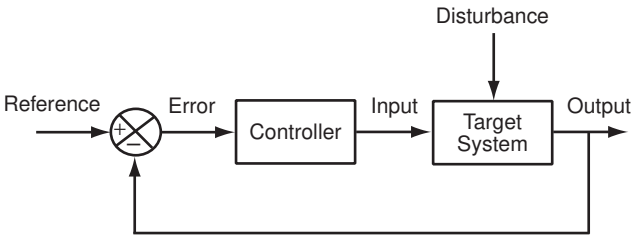


Fig. 1. Block diagram for a simplistic control loop [6].

In short, the design of control functions is based on approximated models (a perfect model is not necessary, nor is it always feasible) of the dynamics of the process to be controlled (*Target System* in Figure 1), in order to derive controllers with guaranteed concerning convergence, avoidance of oscillations, mastering of overshoot effects, etc. This process implies the identification of certain variables of interest in the infrastructure:

- The *output*, which is a measurable indicator of the state of the system.
- The *reference*, the objective value to which we want to drive the *output*.
- The *input*, the action through which the *output* can be modified or regulated to match the *reference* value.
- The *error*, the difference between the *reference* value and the system's *output*, as a measure of deviation.
- The *disturbance*, an external agent that affects the system's dynamics in an unpredictable way.

Through the knowledge of a system's dynamical model, control theory approaches can deliver management strategies that define how to adjust system's inputs to get the desired outputs, i.e. to control the state of the system. Traditional control solutions include Proportional-Integral-Derivative (PID) controllers [7]: the value of the commands to be executed is given by an equation with three terms: $P$ proportional to the *error*, $I$ involving an integration over time of its past values (i.e., a memory-like effect), $D$ based on its current "speed" or rate of change, which takes into account possible future trends. More advanced approaches incorporate the use of optimal control theory, where management strategies are derived from the formulation of objective functions (so-called cost functions) that describe the goals of the administration loop as a mathematical expression. This scope enables multi-objective control schemes when dealing with infrastructures composed of several inputs and outputs, but requires a fine-tuning of the objective function, and a fairly accurate knowledge of the dynamical model of the system.

HPC scheduling is a problem that has already been addressed by means of Control Theory in the past for specific objectives. It has been applied for achieving predictable scheduling by ensuring each job in a workflow to meet its deadline, and additionally regulating incoming jobs through admission control [8], but the proposed solution assumes each job's progress can be measured by the algorithm, which is not always true in HPC environments. More recent works use

model predictive control (MPC) to increase cluster utilization while minimizing slack in periodic workflows [9], [10], but they assume to count on *a priori* knowledge of the job's duration (that should be provided by the user), leading to a control scheme that is not robust to deviations from the analyzed workflows. In any case, the HPC scene has still plenty of room for innovation in terms of autonomy.

### B. Overall Architecture of the system

*1) A lightweight grid: CiGri:* CiGri[1] is a simple, lightweight, scalable and fault tolerant grid system which exploits the unused resources of a set of computing clusters, a concept used in similar solutions such as OurGrid [11] and Condor [12]. This grid is based on two software tools. The CiGri server software is based on a database and offers a user interface for launching grid computations (scripts and web tools). It interacts with the computing clusters through the Ressource Job Management System OAR [13], a batch scheduler software. This lightweight grid system is a simplified version of the general grid concept [14], in particular a certain homogeneity of services and administration procedures are adopted. Also, this grid system is well adapted to address bag-of-tasks workload which is composed of lots of independent and identical tasks. The nature of this specific kind of workload arises from the need of running large-scale scientific computations with different input parameters, by means of grid computing infrastructures. An example of this implementation is MCFOST [15], a 3D radiative transfer code based on Monte-Carlo method, which can usually amount to 30000 tasks with average computation times of 45 minutes per job. Typically, parametric workloads of this kind show small variability in their execution time. According to our analysis of historical data from CiGri database, distribution of computation times exhibits coefficients of variability [16] that ranges from 10% to 30%, indicating negligible dispersion. Sample histograms of computation times for different campaigns are shown in Figure 2.

Jobs in CiGri bag-of-tasks are essentially treated as low-priority tasks intended to be run on cluster's idle resources. This means that, if during their execution, the resource is requested by a local cluster user, the best-effort job is killed by the local batch scheduler and later resubmitted according to specific fault-treatment mechanisms [17]. Additionally, the notion of best-effort means there is no QoS objective to meet, and so the problem falls outside the classical scheduling theory. As a consequence, there are aspects -such as the queued time of a job- that are excluded from the analysis. To sum up, the challenge of CiGri is to guarantee the complete execution of this scientific computations in despite of resources' volatility, in the most efficient way. This clearly implies maximizing the amount of idle resources used by best-effort jobs, without overloading any component of the infrastructure. Figure 3 presents the overall architecture of the CiGri lightweight grid

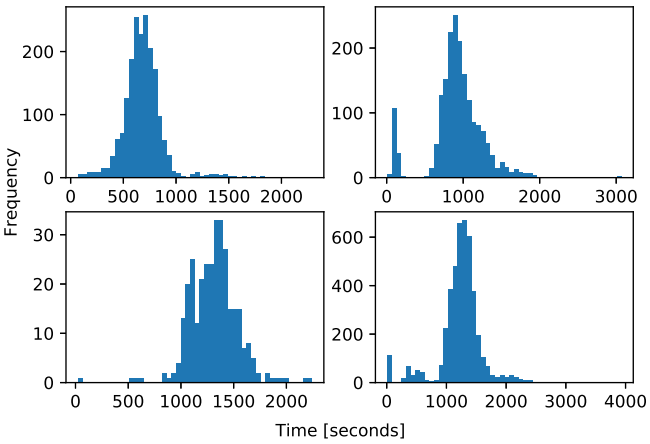


Fig. 2. Campaign histograms for different types of workloads handled by CiGri.

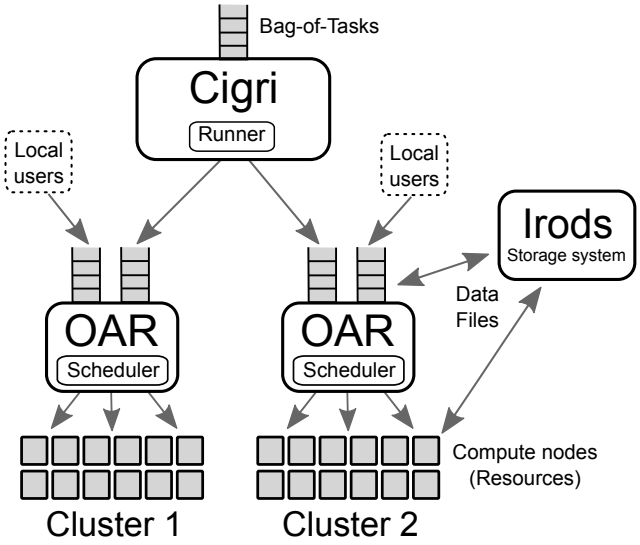


Fig. 3. The system global architecture.

with the OAR batch scheduler and the Irods storage system [18].

There are two options for job submission:

- job submission by a user, directly through OAR.
- submission of a jobs campaign through CiGri: with numerous small jobs, managed in a best-effort manner.

The downloading and uploading of input data and results of the jobs is handled by *iRODS*, a distributed storage management system.

*2) Computing infrastructure:* CiGri production environment is currently deployed onto CIMENT, part of the Gricad unit[2] and one of the most powerful HPC tier-2 centers in France, located at Grenoble. It is a joint center from the Community Université Grenoble Alpes and more than 30 research laboratories from CEA, INRIA and CNRS; composed of over 6500 cores that add up to 135 TFlop/s and 28 TB of

[1]http://ciment.ujf-grenoble.fr/CiGri/dokuwiki/doku.php

[2]https://gricad.univ-grenoble-alpes.fr/

memory distributed over 12 clusters. Its resources are managed by OAR[3], a modular batch scheduler for HPC clusters that supports integration with CiGri through a dedicated waiting queue for best-effort jobs introduced in each cluster. Even though the center is composed of heterogeneous resources, they are grouped by similar specifications (i.e. processing power) to achieve certain uniformity throughout clusters, so that resource homogeneity can be assumed.

The Gricad unit provides HPC resources to academic research communities from a wide range of disciplines: Biology and Health, Chemistry, Environment and Climate, Numerical Physics, Earth and Planetary Sciences, and Distributed Computing [19]. As an example, the center has been extensively used in the ATLAS particle detector experiment [20], one of the seven experiments performed at the LHC (Large Hadron Collider) particle accelerator, and one of the two involved in the discovery of the Higgs boson.

*3) Administration problems considered:* The current, *ad hoc*, solution to manage job submission is based on a *tap* mechanism implemented in the Runner component of CiGri. It maps jobs from the bag-of-tasks on the available resources of the cluster with the following criteria,

**Algorithm II.1:** JOB SUBMISSION( )

**main**
$rate \leftarrow 3$
$increase\,factor \leftarrow 1.5$
**while** $jobsleftinthebag - of - tasks$
**do** $\begin{cases} launchrateamountofjobs \\ \textbf{while} jobsrunning > 0 \textbf{ and } not(timeout) \\ \quad \textbf{do } sleep \\ rate \leftarrow \min(rate * increase\,factor, 100) \end{cases}$

The algorithm cyclically submits *rate* number of jobs to the cluster, which increases at every iteration. Note that, in each cycle, the algorithm waits for the completion of all submitted jobs, but taking into account several failsafe mechanisms intended to protect the infrastructure. As an example, the loop timeouts when job's walltime is exceeded. Other more complex scenarios are also considered by the algorithm, which might involve cluster *blacklisting* depending on the severity of the problem, requiring administrator intervention to regularize the situation.

Under the explained behavior, jobs are submitted onto a cluster only when its waiting queue is fully empty. In practice, this behavior yields situations where the cluster is being underused (or not used at all) in spite of the existence of remaining jobs in the CiGri bag-of-tasks. This occurs when, between cycles of the algorithm, there is no queued jobs.

Other side problems that can occur are:

- Cluster overload, when too many jobs are submitted, and can not be served by the cluster resources: a *stress factor* is defined, between 0 and 1, and sent back to CiGri, so that it avoids sending jobs to a cluster for which it is beyond 0,8.
- Irods overload, when too many requests are submitted.
- Particular storage resource overload, concerning access to an individual storage server.

As a result, the way CiGri handles jobs submission to the waiting queue turns out to be a major factor in the overall performance. Maximizing the exploitation of idle resources, then, requires an improved strategy in this regard.

## III. ANALYSIS AND MODELING

As said before, in this paper we focus on the problem of minimizing the unused resources of a grid while avoiding overloading. To do this, we followed this methodology,

1) informal analysis of the system in order to extract the feedback loop: identify sensors and actuators (which are the inputs and outputs of the process) and formulate the control objective.
2) open loop testing to approximate an analytic model, by measures in the absence of regulation, in order to acquire some parameters of the system behavior.
3) design of the control law at the heart of the Autonomic Manager: in this case simple P and PI regulators.

### A. The considered administration loop

Figure 4 sketches the process detailed in Section II-B3, highlighting elements involved in our administration loop:

- *The Cluster* is the main component: we want to control the "pressure" it undergoes (risk of overload).
- *The Resource Job Management System (RJMS)* is done by OAR which plans execution of jobs on the cluster.
- *The "Runner" module* : submits CiGri jobs as presented in Section II-B3.

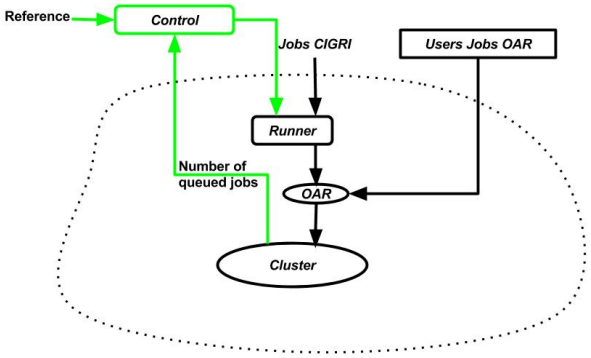


Fig. 4. Feedback loop for the CiGri cluster

Moreover, Figure 4 shows in green the feedback loop we intend to construct. Our intention is to regulate around a fixed number of jobs in the queue waiting to be executed. At runtime, we cyclically measure the number of jobs in the queue and compare this value with a reference value of number of jobs in the queue. Based on this difference, our administration loop (the controller) will automatically decide how much to

[3]http://oar.imag.fr/

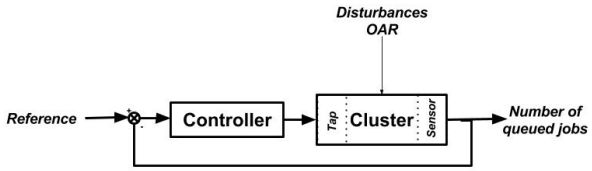


Fig. 5. Diagram of the model-based feedback control loop.

open the tap in order to send more or less CiGri jobs through the Runner to the Cluster. This type of control will allow us to maximize the number of resources utilized by the cluster without overloading it.

We consider that users can also send jobs to the Cluster directly through OAR, which represents an unknown and non-measurable disturbance to our system. These jobs have a non-negligible impact on the cluster charge, therefore on its job treatment speed, and finally on the number of jobs left in the queue.

Figure 5 presents the feedback loop from a Control Theory perspective. We can notice on this figure the actuator (the *tap*), the sensor which measures the number of jobs in the queue, as well as the external perturbations which act on the system: the users jobs sent directly through OAR.

Since the sensor measuring the number of jobs in the queue was not present on our system, we had to construct it. In order to do this, we placed in the core of the Runner module the instructions detailed in Code 1. Line 1 gets the number of jobs on a cluster, and Line 3 gets the jobs in the queue and therefore counts their number.

```
1 cluster_jobs = cluster.get_jobs()
2 temps = Time.now
3 jobs = cluster_jobs.select{ |j| j["state"] == "Waiting" }
4 file = File.open("/home/user1/Nb_jobs.txt", "a+")
5 file << "#{jobs.length};#{temps.to_f}\n"
6 file.close
```

Listing 1. Sensor measuring the number of jobs in the queue

Since the administration loop is incorporated into the main cycle of the Runner, it will also run periodically and with the same frequency. Although this cycle time might vary during the execution depending on the amount of workload the Runner is under, it is lower bounded to 15 seconds. In practice, there are no significant deviations from this bound, so we assume it to be constant.

Another key factor in the behavior of the loop is the selection of the reference value for the waiting queue, in which there is a clear trade-off. A lower number of jobs in the waiting queue might lead to cases with free resources in the OAR cluster, but no available jobs to run, which translates into under-usage of idle resources. This effect can be even worse in the presence of disturbances: user jobs not managed by CiGri, when finished, release a big number of resources that can be potentially used by best-effort jobs, so having a considerable amount of waiting jobs ready to be launched is a good way to deal with this uncertainties. However, an excessively large number of queued jobs can certainly overload

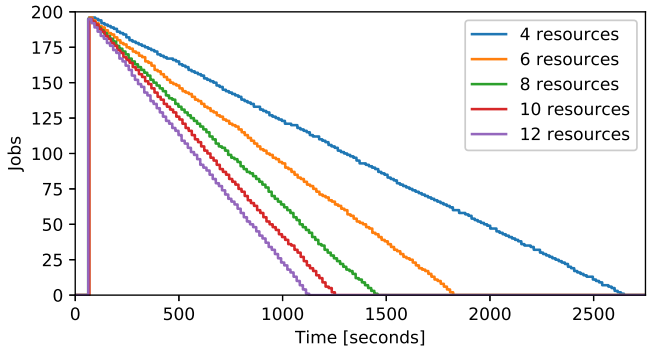


Fig. 6. Number of jobs in the queue for varying resource numbers and fixed job campaign of 200 jobs. Campaign start time among experiments is synchronized to the same time instant $t = 80s$ for better comparison of the responses.

the OAR scheduler. For this reasons, the selection of this value is not straightforward: to our understanding, it should be dynamically adjusted to meet the administration loop's objectives in an optimal way. For this first approach, we manually adjusted the reference to 40 waiting jobs, as this value delivered reasonable results throughout our experiments.

### B. Open loop testing for model identification

In order to have an idea about the behavior of the system, we started with some open loop tests, that is to say the system is left free to evolve while only monitoring its behavior. We test the behavior of the system once when resource number is fixed (jobs are varying) and once when the initial number of jobs is fixed (number of available resource is varying).

For the experiments, we used workflows composed of dummy jobs with distribution of execution times similar to those analyzed in the previous section (Figure 2) to ensure a representative scenario. In particular, the mean execution time for this experiment was 30 seconds.

The evolution of the queue when the initial number of jobs is fixed (200 jobs) and when the number of resources varies is given in Figure 6. As expected, the number of available resources has an impact on the evolution of the number of jobs in the queue: the larger the number of resources, the steeper is the slope of the queue drain.

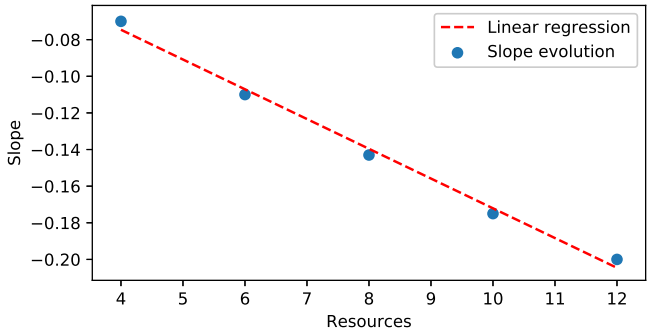


Fig. 7. Slope evolution towards available resources

After testing the queue behavior when the number of resources is fixed and with varying initial number of jobs in the queue, we obtained a relation between the slope of the draining queue and the number of available resources, as depicted in Figure 7. Moreover, we can notice a linear evolution of the slope towards resource number.

### C. A Proportional controller

As a controller in Figure 5, we use a proportional one written: $tap = K_p * error$, where $K_p$ is the gain of the controller. In this case, the error between the number of jobs set as reference and the actual number of jobs is multiplied by a fixed gain, then sent to the *tap*. The choice of this gain is not straightforward as a small value will imply a slow filling of the cluster, while a large value could lead to oscillations of the queue size, which at the end can worsen the performance of the system. In our case, for a fixed number of resources (see Figure 6), we approximate the system behavior as a second order differential equation expressed through its Laplace transformation (a well documented method in control theory [7]) as,

$$H(p) = 1 - \frac{K_p}{(1 + T_1\,p)(1 + T_2\,p)},$$

with $T_1 = 125$ and $T_2 = 235$. The values of $T_1$ and $T_2$ are computed using the Identification Toolbox in Matlab or the well known Cadwell technique [7]. Based on the simulations on the identified (approximated) system, we choose a Proportional controller which is not very aggressive to the system and does not produce overshoot and instability. Using Matlab, we find a suitable value of $K_p = 0.713$. Another gain can be used and the regulation will still work but the system might experience other side problems: lower values can lead to very slow response to disturbances, meaning that the algorithm may take more time to control the queue length in the presence of local user's jobs. By contrast, if the gain is very large, the system can became *unstable*, an undesired property of dynamical systems that causes the queue length to grow unbounded instead of remaining steady as desired.

### D. A PI controller

In a second stage, we designed a PI controller by extending the obtained P controller from previous section to accomplish integral action. In other words, the new control law as a function of time becomes,

$$tap(t) = K_p\,error(t) + K_i \int_0^t error(\tau) d\tau.$$

This expression describes the general form of a PI controller in continuous time, but the implementation of such controller requires a time-discrete control law, which can be done by approximating the integral of the error as a cumulative sum of the previous values of this error function,

$$tap(t) = K_p\,error(t) + K_i \sum_{k=0}^{t} error(k)\Delta t_k,$$

where $\Delta t_k$ corresponds to the cycle time of the main loop in the Runner module at time $k$. This expression shows the main effect of PI controllers: a control action that depends not only of error values computed at present time, but also of error values computed in the past. In practice, it is not necessary to store the value of the $error$ function at each cycle as long as the cumulative error is kept,

```
err = job_reference - queue_load # Compute current error
errcum = errcum + err # Compute cumulative error
tap = kp*err + ki*errcum*dt # Compute control action
```

Listing 2. Pseudo-code for the PI implementation

The proportional gain $K_p$ is tuned as in the P controller. The contribution from the integral term depends on the cumulative error computed at each time instant, but also on the amount of time the error persisted. This effect should be taken into consideration when tuning $K_i$ parameter to avoid overshooting in the time response of the controller. In the control theory framework, overshoot is the effect in which the controlled variable exceeds its target, a common issue that occurs when cumulating "too much" in the integral term ($errcum$ for this case). A frequent workaround to this problem is to implement an anti-windup scheme [21]. For this particular case, we imposed lower and upper thresholds on $errcum$ (of 20 and $-20$ respectively) to limit the effect of the integral action term. Another key characteristic of the integral action is that it accomplishes null steady-state error, in contrast with plain $P$ controllers. This basically means that the amount of jobs in the queue will asymptotically tend to the desired value (and so the difference between this two variables will be equal to zero).

## IV. Experimental validation

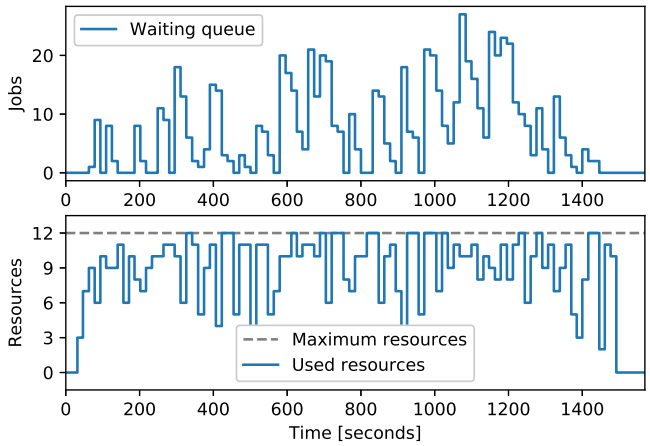


Fig. 8. Open loop behavior (i.e. original algorithm).

In this Section we validate our approach on the real system. The behavior of the system without control is depicted in Figure 8, the P controller is considered in Figure 9 and the PI one can be seen in Figure 10. The behavior of the system through the experiments is characterized by the *waiting*

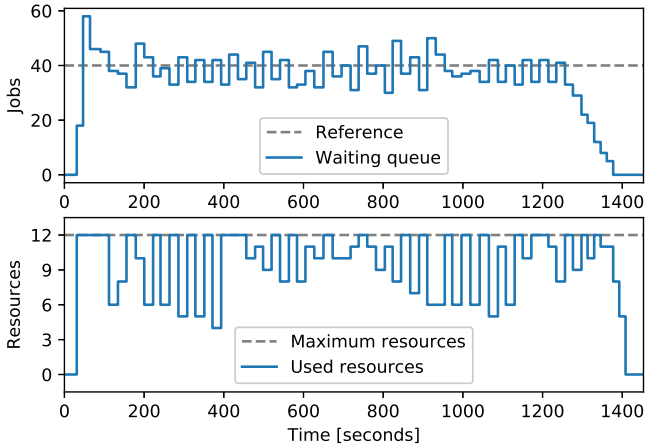


Fig. 9. Closed loop behavior with the P controller.

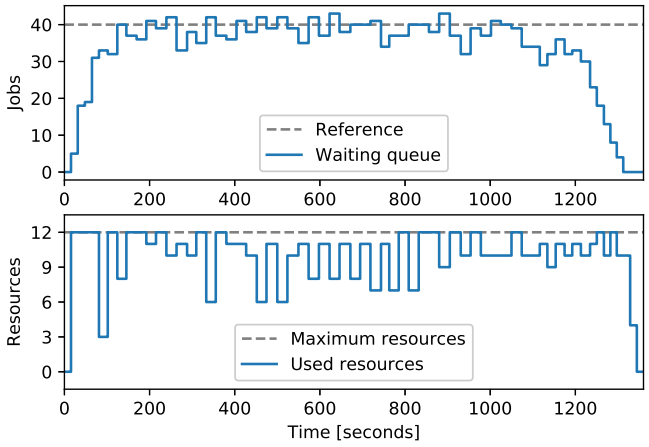


Fig. 10. Closed loop behavior with PI controller.

*queue* (upper graph in Figures) and the *running jobs* (lower graph in Figures). The experiment consisted of a 400 jobs campaign running on a 12 resources cluster. We performed this experiment 10 times per scheme and computed the average performance, depicted in Table I. As in the open loop analysis, we used the same dummy jobs with mean execution time of 30 seconds and small variance, based on real workflows previously processed by CiGri. As we can see, the use of a controller diminishes considerably the completion time of the campaign. This leads to an average improvement in cluster utilization of 5% with the simplest control algorithm, and of 8% in the case of the PI controller. The usage upgrade is reflected in a significant reduction of the total time: approximately 10% faster completion time was achieved for the same campaign.

The cluster usage in an interval $[t_0, t_f]$ was computed using the following expression,

$$Usage[\%] = \frac{1}{T} \sum_{k=t_0}^{t_f} \frac{r(k)\Delta t}{r_{max}}, \quad (1)$$

where $r(k)$ is the number of resources used by best-effort jobs at time $k$ and $r_{max}$ the total amount of resources in the cluster

| | w/o control | P control | PI control |
|---|---|---|---|
| Total time | 1445 sec | 1361 sec | 1313 sec |
| Cluster usage | 77% | 82% | 85% |

TABLE I
COMPARISON OF PERFORMANCE AMONG THE SOLUTIONS FOR THE SAME CAMPAIGN OF 400 JOBS IN A 12 RESOURCES CLUSTER.

(constant, in this case). The total time of the experiment is computed as $T = t_f - t_0$.

Moreover, for further evaluating the performance of the solution, we tested the robustness of our control algorithm. In order words, we analyzed the controller performance when dealing with unknown external factors. Specifically, we stressed the system with external disturbances by submitting higher priority jobs to the cluster, in order to dynamically vary the amount of available resources. The results of the controlled system for this scenario are shown in Figures 11 and 12. The periods when the cluster is unutilized are also diminished, even though the system is not able to measure the amount of available resources. The latter varies randomly by occupying resources in a range of 0-12 equiprobably, making the expectancy of resource availability equal to 6 (half of the total resources), so the completion time of the campaign is expected to be twice as much as that of non-disturbed experiments. However, the usage can be still analyzed as in 1 by computing $r_{max}$ in terms of the time-varying function $r_{available}(k)$,

$$r_{max} = \frac{1}{T} \sum_{k=t_0}^{t_f} r_{available}(k)\Delta t.$$

Needless to say, computing the cluster usage is equivalent to compute the ratio of the area below the *used resources* curve to the area below the *available resources* curve. From the experiments, we observed an average cluster usage of 79% in the presence of disturbances using the $PI$ controller.

It is noteworthy that the behavior of the system may vary when dealing with workflows other than those used for experimentation. However, as one of the advantages of control theory approaches, the controller will certainly ensure the waiting queue to converge to the reference value in any scenario, in despite of this variations.

## V. CONCLUSION AND PERSPECTIVES

*1) Results:* we presented preliminary results addressing autonomic administration in HPC systems, more particularly automated resource management using techniques from Control Theory. We focused on a controller that maximizes cluster utilization while avoiding overload. For this, we performed a control-oriented analysis of the system involving the identification of its dynamics, taking into account the overload problems to be tackled. Further on, we proposed two feedback controllers based on P (Proportional) and PI (Proportional+Integral) regulation; we implemented the approach and validated it through experimental results. Comparing with the previously used, more *ad hoc* solution, the new controller proved to be more efficient in terms of usage of idle resources.

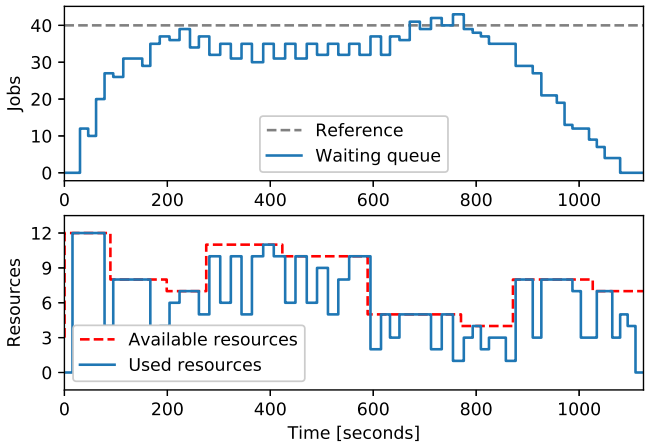


Fig. 11. Test in presence of perturbations in the amount of available resources (40 jobs reference) for a 200 jobs campaign.

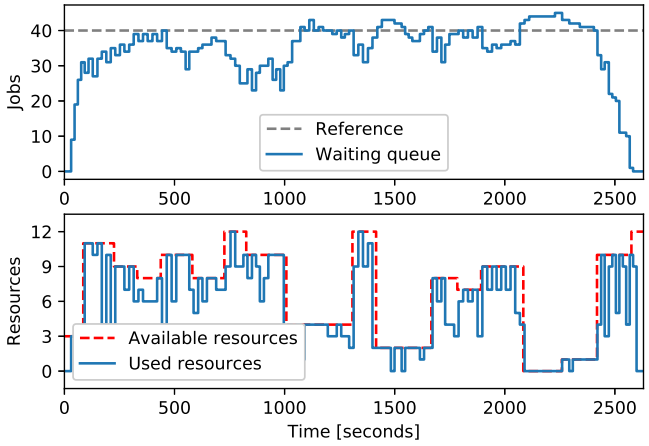


Fig. 12. Test in presence of perturbations in the amount of available resources (40 jobs reference) for a 400 jobs campaign.

The main contribution of this work is a proof of concept. We showed, for a particular case study, the benefits of rethinking computing infrastructures as dynamical systems, to allow the use of the control theory toolset in order to solve specific problems and improve performance measures of computing systems. In past works [9], [10], the identification process was performed taking into account several input and output variables of the infrastructure, which leads to more comprehensive models intended to be used for multi-objective control. However, the obtained linear models lack the ability to reflect several non-linear behaviors and variable constraints of the real system, since its accuracy is limited to a specific *region of operation*. Thus, the performance improvements are constrained to this region, and the controller fails to ensure the desired performance and stability objectives for all possible situations. Our solution do not seek to replace the scheduler but to complement it, which simplifies the whole approach, simultaneously ensuring robustness to variations in the analyzed scenarios.

*2) Perspectives:* We have ongoing work on considering other problems of overload concerning the storage architecture. On the control side, we plan to consider more elaborate techniques which made their proof when applied to other computing systems, e.g. adaptive [22], discrete [23], or stochastic. Another promising research direction is to adopt an integral point of view of the infrastructure. In practice, the running module manages each cluster submissions separately. A centralized administration loop could provide an improved strategy towards a unified framework for autonomic grid management.

## References


[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *IEEE Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[2] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*. Wiley-IEEE, 2004.

[3] M. Litoiu, M. Shaw, G. Tamura, N. M. Villegas, H. Müller, H. Giese, R. Rouvoy, and E. Rutten, "What Can Control Theory Teach Us About Assurances in Self-Adaptive Software Systems?" in *Software Engineering for Self-Adaptive Systems III. Assurances.*, ser. Lecture Notes in Computer Science, R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, Eds. Springer, Jan. 2018, vol. 9640.

[4] E. Rutten, N. Marchand, and D. Simon, "Feedback Control as MAPE-K loop in Autonomic Computing," in *Software Engineering for Self-Adaptive Systems III. Assurances.*, ser. Lecture Notes in Computer Science, R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese, Eds. Springer, Jan. 2018, vol. 9640. [Online]. Available: https://hal.inria.fr/hal-01285014

[5] M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand, and B. Robu, "Feedback autonomic provisioning for guaranteeing performance in mapreduce systems," *IEEE Transactions on Cloud Computing*, 2016.

[6] T. Abdelzaher, Y. Diao, J. L. Hellerstein, C. Lu, and X. Zhu, "Introduction to control theory and its application to computing systems," in *Performance Modeling and Engineering*. Springer, 2008, pp. 185–215.

[7] W. S. Levine, *The Control Handbook, Second Edition (three volume set)*. CRC Press, 2010.

[8] S.-M. Park and M. Humphrey, "Predictable high-performance computing using feedback control and admission control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 396–411, 2011.

[9] H. A. Ghazzawi, I. Bate, and L. S. Indrusiak, "A control theoretic approach for workflow management," in *Engineering of Complex Computer Systems (ICECCS), 2012 17th International Conference on*. IEEE, 2012, pp. 280–289.

[10] ——, "Mpc vs. pid controllers in multi-cpu multi-objective real-time scheduling systems," in *Proceedings of The*, 2012, pp. 77–83.

[11] W. Cirne, F. Brasileiro, N. Andrade, L. B. Costa, A. Andrade, R. Novaes, and M. Mowbray, "Labs of the world, unite!!!" *Journal of Grid Computing*, vol. 4, no. 3, pp. 225–246, 2006.

[12] A. R. Butt, R. Zhang, and Y. C. Hu, "A self-organizing flock of condors," *Journal of parallel and distributed computing*, vol. 66, no. 1, pp. 145–161, 2006.

[13] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard, "A batch scheduler with high level components," in *IEEE Int. Symp. on Cluster Computing and the Grid, CCGrid*, 2005, pp. 776–783.

[14] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[15] C. Pinte, F. Ménard, G. Duchêne, and P. Bastien, "Monte carlo radiative transfer in protoplanetary disks," *Astronomy & Astrophysics*, vol. 459, no. 3, pp. 797–804, 2006.

[16] R. Bendel, S. Higgins, J. Teberg, and D. Pyke, "Comparison of skewness coefficient, coefficient of variation, and gini coefficient as inequality measures within populations," *Oecologia*, vol. 78, no. 3, pp. 394–400, 1989.

[17] Y. Georgiou, O. Richard, and N. Capit, "Evaluations of the lightweight grid cigri upon the grid5000 platform," in *e-Science and Grid Computing, IEEE International Conference on*. IEEE, 2007, pp. 279–286.

[18] A. Rajasekar, R. Moore, C.-y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert *et al.*, "irods primer: integrated rule-oriented data system," *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 2, no. 1, pp. 1–143, 2010.

[19] C. Biscarat and B. Bzeznik, "Synergy between the ciment tier-2 hpc centre and the hep community at lpsc in grenoble (france)," in *Journal of Physics: Conference Series*, vol. 513, no. 3. IOP Publishing, 2014, p. 032008.

[20] G. Aad, E. Abat, J. Abdallah, A. Abdelalim, A. Abdesselam, O. Abdinov, B. Abi, M. Abolins, H. Abramowicz, E. Acerbi *et al.*, "The atlas experiment at the cern large hadron collider," *Journal of instrumentation*, vol. 3, no. 8, pp. S08 003–S08 003, 2008.

[21] A. T. Azar and F. E. Serrano, "Design and modeling of anti wind up pid controllers," in *Complex system modelling and control through intelligent soft computations*. Springer, 2015, pp. 1–44.

[22] S. Cerf, M. Berekmeri, B. Robu, N. Marchand, S. Bouchenak, and I. Landau, "Adaptive feedforward and feedback control for cloud services," in *20th World Congress of the International Federation of Automatic Control (IFAC 2017)*, Jul 2017.

[23] N. Berthier, F. Alvares, and E. R. Gwenaël Delaval, Hervé Marchand, "Logico-numerical control for software components," in *1st IEEE Conf. on Control Technology and Applications, CCTA*, Hawai'i, USA, 2017.